

Chebyshev polynomial filtered subspace iteration in the Discontinuous Galerkin method for large-scale electronic structure calculations

Amartya S. Banerjee,^{1, a)} Lin Lin,^{2, 1, b)} Wei Hu,^{1, c)} Chao Yang,^{1, d)} and John E. Pask^{3, e)}

¹⁾ *Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA*

²⁾ *Department of Mathematics, University of California, Berkeley, CA 94720, USA*

³⁾ *Physics Division, Lawrence Livermore National Laboratory, Livermore, CA 94550, U.S.A*

(Dated: September 15, 2016)

The Discontinuous Galerkin (DG) electronic structure method employs an adaptive local basis (ALB) set to solve the Kohn-Sham equations of density functional theory (DFT) in a discontinuous Galerkin framework. The adaptive local basis is generated on-the-fly to capture the local material physics, and can systematically attain chemical accuracy with only a few tens of degrees of freedom per atom. A central issue for large-scale calculations, however, is the computation of the electron density (and subsequently, ground state properties) from the discretized Hamiltonian in an efficient and scalable manner. We show in this work how Chebyshev polynomial filtered subspace iteration (CheFSI) can be used to address this issue and push the envelope in large-scale materials simulations in a discontinuous Galerkin framework. We describe how the subspace filtering steps can be performed in an efficient and scalable manner using a two-dimensional parallelization scheme, thanks to the orthogonality of the DG basis set and block-sparse structure of the DG Hamiltonian matrix. The on-the-fly nature of the ALBs requires additional care in carrying out the subspace iterations. We demonstrate the parallel scalability of the DG-CheFSI approach in calculations of large-scale two-dimensional graphene sheets and bulk three-dimensional lithium-ion electrolyte systems. Employing 55,296 computational cores, the time per self-consistent field iteration for a sample of the bulk 3D electrolyte containing 8,586 atoms is 90 seconds, and the time for a graphene sheet containing 11,520 atoms is 75 seconds.

I. INTRODUCTION

Kohn-Sham Density functional theory (KS-DFT)^{1,2} is the most widely used methodology for electronic structure calculations of condensed matter and nano-material systems. KS-DFT requires the solution of a nonlinear eigenvalue problem, and this is usually achieved by means of self-consistent field (SCF) iterations in conjunction with convergence acceleration schemes^{3,4}. The most computationally intensive part of conventional KS-DFT calculations is the solution of the linear eigenvalue problem associated with diagonalization of the Kohn-Sham Hamiltonian on every SCF step. The results of this eigenvalue problem are used to update the electron density ρ , from which the various terms of the Kohn-Sham Hamiltonian are computed. As the SCF iterations progress, the solution to the linear eigenvalue problem on successive SCF steps forms increasingly better approximations to the actual Kohn-Sham eigenstates.

The computational complexity (or algorithmic complexity) of the solution of the linear eigenvalue problem, with respect to the number of electronic states in the system, is dependent on the algorithm used for solution

of the eigenvalue problem – in particular, it depends on whether direct or iterative methods of solution are used. The prefactor in such algorithmic complexity estimates strongly influences the simulation wall times in practical computations. The ability to tackle large-scale complex materials science problems, therefore, is closely related to how small the prefactor can be made in real computations, regardless of which diagonalization algorithm is used.

The prefactor is not only influenced by the choice of algorithm, but also by the discretization scheme. Specifically, it depends on the number of basis functions per atom required to obtain accurate and reliable results. The widely used planewave method^{3,5,6} allows high fidelity calculations to be carried out since systematic convergence with respect to the number of basis functions per atom can be obtained. However, this method requires a large number of basis functions per atom – often thousands or more planewaves per atom need to be employed. Similar observations hold true for methods based on finite elements^{7–10}, finite differences^{11–14}, and other planewave-like spectral basis functions^{15,16}. On the other hand, methods based on atom centered basis functions^{17–20} typically require fewer basis functions per atom (often, as few as 10 – 80). However, it can be nontrivial to improve the quality of solutions obtained via such methods, as a result of which the success of a practical calculation can depend on the experience of the practitioner.

In a series of recent contributions^{21–24}, a new method-

^{a)} asb@lbl.gov

^{b)} linlin@math.berkeley.edu

^{c)} whu@lbl.gov

^{d)} cyang@lbl.gov

^{e)} pask1@llnl.gov

ology for discretizing the Kohn-Sham equations using so called adaptive local basis functions (ALBs) has been presented. This approach involves partitioning the global simulation domain into a set of subdomains (called elements) and solving the Kohn-Sham equations locally in and around each element. The results of these local calculations are used to generate the ALBs (in each element) and the Kohn-Sham equations in the global simulation domain are then discretized using them. Since the ALBs form a discontinuous basis set globally (the discontinuity occurs at the element boundaries), the interior penalty Discontinuous Galerkin (DG) approach²⁵ is used for constructing the Hamiltonian matrix. The DG formulation ensures that the global continuity of the relevant Kohn-Sham eigenstates and related quantities such as electron density is approximately obtained.

The solution obtained by the above procedure converges systematically to the infinite basis set limit as the number of ALBs is increased. The error in this scheme can be gauged by means of *a posteriori* error estimators^{26–28}. Owing to the fact that the ALBs incorporate local materials physics into the basis, an efficient discretization of the Kohn-Sham equations can be obtained in which chemical accuracy in total energies and forces can be attained with a few tens of basis functions per atom^{21,22}. The DG approach for solving the Kohn-Sham equations with ALBs has been incorporated into a massively parallel software package called DGDFT^{23,24}.

Although the DG framework for the Kohn-Sham equations (as implemented in the DGDFT code) has been successfully used to study materials problems involving many thousands of atoms²⁴, a persistent issue has been to obtain the electron density from the discretized Kohn-Sham Hamiltonian in an efficient manner for systems containing a thousand atoms or more. Due to the relatively small size of the discretized Hamiltonian matrices involved, direct diagonalization methods (via ScaLAPACK^{29,30}, for example) are feasible for systems of smaller size. However, the computational cost of these methods scales in a cubic manner, i.e., as $\mathcal{O}(N_b^3)$, with N_b denoting the total number of basis functions used in the simulation. Thus, the computational cost increases steeply with respect to the size of the system. The cubic scaling problem is compounded by the fact that direct diagonalization solvers (for dense matrices) typically do not scale well beyond a few thousand processors on distributed memory machines. In recent work with the DGDFT code in massively parallel computing environments^{23,24}, we have found that for systems containing more than a few thousand atoms, the step of obtaining the electron density from the Hamiltonian can consume 95% or more of the total computational time.

Direct diagonalization methods do not take advantage of the fact that the DG Hamiltonian matrix, denoted henceforth by H^{DG} , is a block-sparse matrix. The sparsity of H^{DG} allows several alternatives for mitigating the issues associated with direct diagonalization methods. One alternative is to employ “linear scaling” methods

(see, e.g., review articles^{31,32}), based on direct calculation of truncated density matrices. While linear scaling methods have been very successful for tackling large insulating systems with sizable band gaps^{33–35}, they are less well suited for metallic systems or semiconducting systems with small band gaps. The sparse nature of the H^{DG} matrix also allows for the Pole Expansion and Selected Inversion (PEXSI) technique^{36,37} to be employed for directly computing the electron density and other ground state properties. The computational cost of the PEXSI technique is at most $\mathcal{O}(N_b^3/N_s) \sim \mathcal{O}(N_s^2)$, with N_s denoting the number of Kohn-Sham states, even for metallic systems. The PEXSI technique has excellent parallel scalability^{23,24,38} and has been shown to work well in conjunction with DGDFT while studying two-dimensional materials. However, it becomes more expensive (both in terms of memory and run time) for three-dimensional bulk materials, and has limited ability to make use of good starting guesses (from previous geometry optimization or molecular dynamics steps) to accelerate computations.

With the above considerations in mind, an alternate strategy for reducing the simulation wall time in practical computations is to revert to the usage of an algorithm that scales in a cubic manner with respect to the system size, but to reduce the pre-constant of the algorithm. This includes the use of iterative diagonalization methods such as the Davidson method^{39,40}, conjugate gradient-type methods^{5,41,42}, and residual minimization methods⁶. However, the effectiveness of these schemes relies on the availability of a good preconditioner, which is currently not available for H^{DG} .

In this work, we utilize the technique of Chebyshev polynomial filtered subspace iteration (CheFSI) to address the diagonalization problem in the DG framework and implement it within the DGDFT code. While the CheFSI technique has been utilized with great success by various practitioners working with finite differences, finite elements, and spectral basis sets^{15,43–49}, its application to basis sets resulting in reduced-size Hamiltonian matrices (such as atomic orbital type or adaptive basis sets), with on-the-fly adaptation in particular, has not been considered before to our knowledge.

The DG framework has a number of features that make the use of CheFSI attractive. First, since the ALBs are orthonormal, one does not need to consider the overlap matrix (for usual pseudopotential calculations), thus ensuring that the CheFSI method in its original form can be readily employed. Secondly, compared to Hamiltonian and overlap matrices resulting from other high-quality orbital-based basis sets, such as augmented Gaussians⁵⁰ or partition-of-unity finite-elements⁵¹, the H^{DG} matrix has relatively small spectral radius – of the order of a few thousand (atomic units) for the systems considered here. As a result of this, Chebyshev polynomial filters of relatively low order suffice. In contrast to direct diagonalization methods which scale as $\mathcal{O}(N_b^3)$, the computational complexity of CheFSI⁴³ is reduced to $\mathcal{O}(N_b N_s^2 + N_s^3)$.

Since N_b/N_s is typically $\sim 2-20$ for ALBs, this reduction of prefactor can be sizable for systems with thousands of atoms, leading to substantially shorter simulation times. Finally, due to the sparse nature of the H^{DG} matrix and its nearest neighbor block structure, the computation of the product of this matrix with a block of dense vectors can be carried out with relatively low communication volume between processors. This observation leads to an efficient and scalable Hamiltonian matrix times vector product implementation that is crucial for the success of the CheFSI method within the DG framework.

Overall, we seek an approach for conventional KS-DFT calculations of large systems with substantially reduced prefactor, while retaining the accuracy, systematic improvability, and general applicability of established planewave and other spectral approaches. This is made possible by the use of ALBs which ensure that the number of degrees of freedom per atom is kept low, combined with the use of the CheFSI method which is known to have a low prefactor compared to conventional algorithms – as long as an efficient implementation of the Hamiltonian matrix times vector product can be set up. As we show subsequently, through this combination of strategies, we are able to tackle systems containing thousands of atoms routinely, with wall times on the order of a few tens of seconds per SCF step on large-scale parallel computing clusters.

The rest of the paper is structured as follows. In Section II, we outline the background on the DG formulation of KS-DFT and the CheFSI method, before delving into the implementation of the CheFSI method within the DG framework. In Section III, we present results and comparisons with competing methods. We conclude and comment on future research directions in Section IV.

II. METHODOLOGY

A. Discontinuous Galerkin formulation of KS-DFT

In this section, we discuss aspects of the DG framework for the Kohn-Sham equations – as implemented within the DGDFT code – relevant to the implementation of the CheFSI method. More details on the theoretical underpinnings and practical implementation strategies of the DG framework can be found in^{21–23}.

In the present work, we consider Γ -point calculations of periodic systems, as typical in *ab initio* molecular dynamics, and large-scale calculations generally. The Kohn-Sham orbitals can be taken to be real valued in this case. In the DG framework, the global simulation domain Ω is partitioned into a number of subdomains (or *elements*) such that the union of these subdomains tile the whole domain and adjacent subdomains are non-overlapping (except possibly at corners, edges, or at a surface). Due to the periodic boundary conditions on Ω , each surface of each element is shared between two neighboring elements. We denote the collection of the sub-domains

as $\mathcal{T} = \{E_1, \dots, E_M\}$ and the collection of all the surfaces as \mathcal{S} . Each element E_K is embedded into a slightly larger *extended element* Q_K that includes a buffer region surrounding E_K . Figure 1a shows a model 2D system partitioned using 16 equal elements $\{E_1, \dots, E_{16}\}$.

Due to the decomposition of Ω into elements, global L^2 inner products between various quantities (denoted here as $\langle \cdot, \cdot \rangle_{\mathcal{T}}$), can be taken as the sum of local L^2 inner products over individual elements. We introduce the notation $\langle \cdot, \cdot \rangle_{\mathcal{S}}$ to define the sum of local L^2 surface inner products on all surfaces of all elements. We will also employ the notation $\{\{ \cdot \}\}$ to denote the average of a quantity across a surface while $[[\cdot]]$ denotes the jump across a surface for discontinuous quantities.

In DGDFT, each SCF iteration includes a preliminary step of generating the ALBs on the fly. This is accomplished by iteratively solving a local Kohn-Sham problem on each of the extended elements – the effective potential used for this calculation is simply the restriction of the effective potential on the global simulation domain to the extended element. The resulting (approximate) Kohn-Sham states over Q_K are then restricted to E_K and orthonormalized to produce the ALBs over E_K .

At the end of the ALB generation process, each element E_K has a collection of J_K ALBs, denoted by $\{\varphi_{K,j}\}_{j=1}^{J_K}$. Each ALB is compactly supported on one element. The complete collection of ALBs

$$\mathcal{A} = \{\varphi_{K,j}\}_{K=1, j=1}^{K=M, j=J_K}, \quad (1)$$

forms an L^2 orthonormal set over Ω , i.e.,

$$\langle \varphi_{K,j}, \varphi_{K',j'} \rangle_{\mathcal{T}} = \delta_{K,K'} \delta_{j,j'}, \quad (2)$$

for $K, K' = 1, \dots, M$; $j = 1, \dots, J_K$ and $j' = 1, \dots, J_{K'}$. The global Kohn-Sham states over Ω can be expanded using the ALBs as:

$$\psi_i(x) = \sum_{K=1}^M \sum_{j=1}^{J_K} c_{i,K,j} \varphi_{K,j}(x). \quad (3)$$

Due to the fact that the ALBs are discontinuous over the global domain, whereas the Kohn-Sham states (and related physical quantities such as the electron density) are continuous, the DG framework penalizes discontinuities in these quantities across element surfaces. Accordingly, the electronic free energy of a system with N_s occupied

electronic states is written as:

$$\begin{aligned}
E_{\text{free}}^{\text{DG}}(\{\psi_i, f_i\}) = & \frac{1}{2} \sum_{i=1}^{N_s} 2f_i \langle \nabla \psi_i, \nabla \psi_i \rangle_{\mathcal{T}} + \langle V_{\text{eff}}, \rho \rangle_{\mathcal{T}} \\
& + \sum_{i=1}^{N_s} 2f_i \sum_{I=1}^{N_A} \sum_{\ell=1}^{L_I} \gamma_{I,\ell} |\langle b_{I,\ell}(\cdot - R_I), \psi_i \rangle_{\mathcal{T}}|^2 \\
& - T_{\text{el}} S_{\text{el}}(\{f_i\}) \\
& - \sum_{i=1}^{N_s} 2f_i \langle \{\{\nabla \psi_i\}\}, [\psi_i] \rangle_{\mathcal{S}} \\
& + \alpha \sum_{i=1}^{N_s} 2f_i \langle [\psi_i], [\psi_i] \rangle_{\mathcal{S}}, \quad (4)
\end{aligned}$$

where, $0 \leq f_i \leq 1$ are the electronic occupation numbers (specified via Fermi-Dirac smearing), V_{eff} denotes the effective potential (consisting of local pseudopotential, Hartree, and exchange correlation contributions), the scalars $\gamma_{I,\ell}$ and projector functions $b_{I,\ell}$ correspond to the nonlocal pseudopotential expressed in the Kleinman-Bylander form⁵², and T_{el} and S_{el} correspond to the electronic temperature and electronic entropy, respectively. The quantity α is an adjustable penalty parameter that ensures that Eq. (4) has a well defined ground state free energy.

Using the ALBs to discretize the above expression for the free energy and subsequently minimizing the discretized energy with respect to the expansion coefficients $c_{i;K,j}$ (as well as the occupation numbers f_i), while maintaining the orthonormality constraint on the orbitals, leads us to the discretized version of the Euler-Lagrange equations. This takes the form of the following eigenvalue problem:

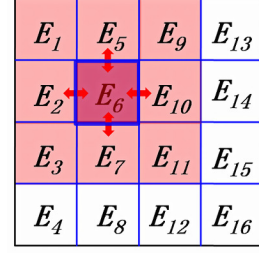
$$\sum_{K',j'} H_{K,j;K',j'}^{\text{DG}} c_{i;K',j'} = \lambda_i c_{i;K,j}, \quad (5)$$

with the discretized Hamiltonian operator expressible as:

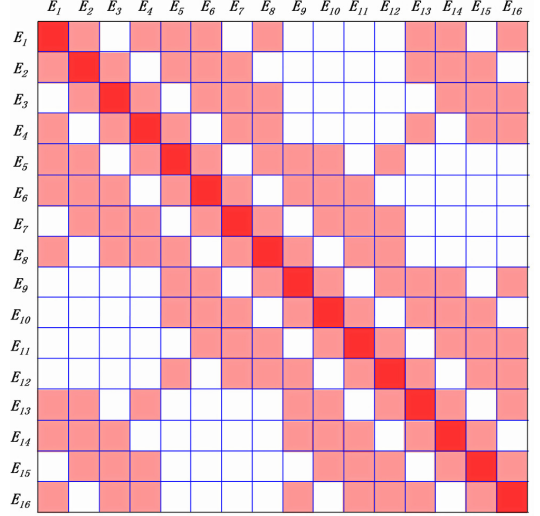
$$\begin{aligned}
& H_{K,j;K',j'}^{\text{DG}} \\
= & \left(\frac{1}{2} \langle \nabla \varphi_{K,j}, \nabla \varphi_{K',j'} \rangle_{\mathcal{T}} + \langle \varphi_{K,j}, V_{\text{eff}} \varphi_{K',j'} \rangle_{\mathcal{T}} \right) \delta_{K,K'} \\
& + \left(\sum_{I,\ell} \gamma_{I,\ell} \langle \varphi_{K,j}, b_{I,\ell} \rangle_{\mathcal{T}} \langle b_{I,\ell}, \varphi_{K',j'} \rangle_{\mathcal{T}} \right) \\
& + \left(-\frac{1}{2} \langle [\varphi_{K,j}], \{\{\nabla \varphi_{K',j'}\}\} \rangle_{\mathcal{S}} \right. \\
& \quad \left. - \frac{1}{2} \langle \{\{\nabla \varphi_{K,j}\}\}, [\varphi_{K',j'}] \rangle_{\mathcal{S}} \right. \\
& \quad \left. + \alpha \langle [\varphi_{K,j}], [\varphi_{K',j'}] \rangle_{\mathcal{S}} \right). \quad (6)
\end{aligned}$$

The matrix H^{DG} can be naturally partitioned into blocks based on the element indices. We will denote the (K, K') -th matrix sub-block (of size $J_K \times J_{K'}$) as:

$$H_{K;K'}^{\text{DG}} = H_{K,j=1,\dots,J_K;K',j'=1,\dots,J_{K'}}^{\text{DG}}. \quad (7)$$



(a) Schematic 4×4 partition of a model 2D computational domain into 16 elements.



(b) The DG Hamiltonian matrix H^{DG} with its block sparsity pattern resulting from such a partition. White represents zero blocks.

FIG. 1. Partitioning of a domain into DG elements and the resulting discretized Hamiltonian H^{DG} .

Since the ALBs are compactly supported on their respective elements, the block $H_{K;K'}^{\text{DG}}$ is non zero only when K and K' refer to neighboring elements. This situation is illustrated in Figure 1.

Further details on interpretation and computation of the various terms described above as well as the significance of the parameter α in practical calculations can be found in^{21,23}. Note that the appearance of average and jump operators in Eq. (4) and Eq. (6) are a distinguishing feature of the interior penalty DG formulation of the Kohn-Sham equations.

During the SCF iterations, the matrix H^{DG} is constructed using the most recent effective total potential V_{eff} . Following this, the electron density needs to be computed from H^{DG} . So far, this step has been achieved in the DGDFT code in two distinct ways. In the first approach, the use of a parallel eigensolver (the ScaLAPACK routine PDSYEV) allows one to directly compute the eigenvalues and eigenvectors of H^{DG} . From these, the orbital occupations $\{f_i\}_{i=1}^{N_s}$ can be computed via Fermi-

Dirac smearing while the density matrix (also called the *Fermi matrix* at finite electronic temperature) and the electron density can be computed from the eigenvectors as:

$$P_{K,j;K',j'} = \sum_{i=1}^{N_s} f_i c_{i;K,j} c_{i;K',j'}, \quad (8)$$

$$\rho(x) = 2 \sum_{K=1}^M \sum_{j=1}^{J_K} \sum_{j'=1}^{J_K} \varphi_{K,j}(x) \varphi_{K,j'}(x) P_{K,j;K,j'}. \quad (9)$$

Eq. (9) shows that only the diagonal blocks of the density matrix need to be computed to evaluate the electron density. Note that the calculation of these blocks can be done individually on each element.

The second approach involves the use of the PEXSI technique to directly compute the density matrix elements, without going through the intermediate eigenvalues and eigenvectors. The electron density can be evaluated subsequently from Eq. (9) using the diagonal blocks of the density matrix, while the computation of forces requires computation of the non-diagonal blocks corresponding to the sparsity pattern of H^{DG} ^{22,38}.

Considering the limitations of the each of the above approaches (as described earlier), we now explore the option of using Chebyshev polynomial filtered subspace iteration to compute the eigenvalues and eigenvectors of H^{DG} .

B. Chebyshev polynomial filtered subspace iteration within DGDFT

Subspace iteration is a generalization of the classical power method for computing the dominant eigenpair of a matrix^{53,54}. The standard subspace iteration can be used to obtain an approximation to the invariant subspace associated with the largest few eigenvalues. In Kohn-Sham DFT, the invariant subspace of interest is the one associated with the occupied states (and possibly a few unoccupied states above the Fermi level)^{55,56} which do not correspond to the dominant eigenvalues of the Kohn-Sham Hamiltonian.

A Chebyshev polynomial $p_m(\lambda)$ can be constructed to map eigenvalues at the low end of the spectrum (corresponding to the occupied states) of H^{DG} to the dominant eigenvalues of $p_m(H^{\text{DG}})$. The exponential growth property of the Chebyshev polynomials outside the region $[-1, 1]$ can be used to ensure that the wanted part of the spectrum (i.e., the occupied states in ground state electronic structure calculations) can be magnified while the unwanted part (corresponding to the unoccupied states) is damped in comparison^{57–59}. Applying subspace iteration to $p_m(H^{\text{DG}})$ yields the desired invariant subspace. Within each iteration, the multiplication of $p_m(H^{\text{DG}})$ with a block of vectors can be carried out by using the three-term recurrence satisfied by Chebyshev polynomials. The application of the Chebyshev polynomial fil-

tered subspace iteration (CheFSI) technique for computing the occupied eigenspace of the Kohn-Sham operator was introduced in^{43,44}. Within the SCF iteration framework, this methodology can be thought of as a form of nonlinear subspace iteration in the sense that the approach de-emphasizes the accurate solution of the intermediate linearized Kohn-Sham eigenvalue problems on every SCF step. With the progress of the SCF iterations, the approximate Hamiltonian approaches the self-consistent one and the span of the (approximately) computed eigenvectors approaches the converged occupied subspace simultaneously. This particular feature of CheFSI has some bearing on the way it is implemented within the DG framework, as explained later.

The main desirable features of CheFSI which make it suitable for application to large-scale electronic structure problems are the following : 1) It is a block method in which H^{DG} can be multiplied with a block of vectors simultaneously. This additional level of concurrency allows the algorithm to achieve better parallel scalability compared to standard Krylov subspace methods such as the Lanczos algorithm. 2) Compared to other Krylov subspace methods, it performs fewer Rayleigh-Ritz calculations in which a projected subspace eigenvalue problem is solved. The Rayleigh-Ritz procedure is often the computational bottleneck when the number of eigenvalues to be computed is relatively large.

The key steps in a CheFSI cycle (see Algorithm 1 for details) are an application of the Chebyshev polynomial filter on a block of vectors, subsequent orthonormalization of the filtered block, a Rayleigh-Ritz step, and finally a so called subspace rotation step^{43,45}. Together, the last three steps will be referred to as solving the *subspace problem*. Note that the Rayleigh-Ritz and subspace rotation steps are useful for explicitly obtaining the (approximate) occupied eigenpairs of the Hamiltonian from the filtered subspace. We will now elaborate on various important aspects of our implementation of these steps within DGDFT.

Algorithm 1 CheFSI cycle

Input: Matrix H^{DG} , starting vector block X , filter order m

1. Compute lower bound b_{low} using previous Ritz values and the upper bound b_{up} using a few steps of the Lanczos algorithm.
2. Perform Chebyshev polynomial filtering, i.e., compute $\tilde{Y} = p_m(H^{\text{DG}})X$ with $[b_{\text{low}}, b_{\text{up}}]$ mapped to $[-1, 1]$.
3. Orthonormalize columns of \tilde{Y} : Set $S = \tilde{Y}^T \tilde{Y}$, compute $U^T U = S$, and solve $\hat{Y} U = \tilde{Y}$.
4. Rayleigh-Ritz step: Compute the projected subspace matrix $\hat{H} = \hat{Y}^T H^{\text{DG}} \hat{Y}$ and solve the eigenproblem $\hat{H} Q = Q D$.
5. Perform a subspace rotation step $X_{\text{new}} = \hat{Y} Q$.

Output: Vector block X_{new} (approximate eigenvectors) and Ritz values D (approximate eigenvalues).

1. The multiplication of H^{DG} with a block of vectors

One of the key computational steps of the CheFSI method is to perform $Y = p_m(H^{\text{DG}})X$. This step requires an efficient and scalable implementation of multiplying H^{DG} with a block of vectors. Additionally, computation of the action of the Hamiltonian matrix on vectors is required for estimating the spectral bounds of the Hamiltonian via the Lanczos algorithm as well as during the Rayleigh-Ritz step (see Algorithm 1).

Given a block of vectors X consisting of columns $\{x_i\}_{i=1}^{N_s}$, the multiplication of H^{DG} with a subset of these columns x_{i_1, \dots, i_2} , with $1 \leq i_1 \leq i_2 \leq N_s$, can be written as:

$$y_{i_1, \dots, i_2; K, j} = \sum_{K', j'} H_{K; K'; j, j'}^{\text{DG}} x_{i_1, \dots, i_2; K', j'}, \quad (10)$$

where, $K, K' = 1, \dots, M$ and $j = 1, \dots, J_K, j' = 1, \dots, J_{K'}$. Using the fact that $H_{K; K'}^{\text{DG}}$ is non-zero only for $K' \in \mathcal{N}(K)$, i.e., the neighboring elements of the K -th element, we may rewrite this as:

$$y_{i_1, \dots, i_2; K, j} = \sum_{K' \in \mathcal{N}(K)} H_{K; K'}^{\text{DG}} x_{i_1, \dots, i_2; K', j'}. \quad (11)$$

Thus, the portion of the resulting set of vectors y_{i_1, \dots, i_2} that is associated with the element K can be written as:

$$y_{i_1, \dots, i_2; K} = \sum_{K' \in \mathcal{N}(K)} H_{K; K'}^{\text{DG}} x_{i_1, \dots, i_2; K'}. \quad (12)$$

Since the individual blocks $H_{K; K'}^{\text{DG}}$ and $x_{i_1, \dots, i_2; K'}$ are dense, Eq. (12) can be computed as the sum of a series of matrix-matrix products (i.e., GEMM operations in Level-3 BLAS). Further, since the above operation can be carried out independently over the various columns of X , it is natural to take advantage of the manifestly parallel nature of the problem by distributing the columns among separate processing elements in an appropriate manner.

The data distribution of the various quantities involved in Eq. (12) is important in deciding how the operation can be carried out in practice. As explained in²³, the DGDFDFT code uses a two level parallelization strategy implemented via Message Passing Interface (MPI) to handle inter-process communication. At the coarse grained level, work is distributed among processors by elements, leading to *inter-element* parallelization. Further, within each element, the work associated with construction of the local portions of the DG Hamiltonian, evaluation of the electron density, and the ALB generation process is parallelized leading to *intra-element* parallelization. The processors are partitioned into a two-dimensional logical process grid with a column major order (Fig 2). We will refer to this layout of the MPI processes as the *global process grid*. For the sake of discussion, we assume here that the total number of MPI processes (in the global process grid) is $P_{\text{tot}} = M \times P_e$, so that there are P_e

$M = \text{Number of DG elements}$

1	P_e+1	$2P_e+1$	\dots	$(M-1)P_e+1$
2	P_e+2	$2P_e+2$	\dots	$(M-1)P_e+2$
3	P_e+3	$2P_e+3$	\dots	$(M-1)P_e+3$
\vdots	\vdots	\vdots	\ddots	\vdots
P_e	$2P_e$	$3P_e$	\dots	MP_e

$P_e = \text{Number of processors per DG element}$

FIG. 2. Two-level parallelization scheme within DGDFDFT : The number of DG elements is M and each element has P_e processors dedicated to it, making the total number of processors $P_{\text{tot}} = M \times P_e$.

processes assigned to each of the M elements. Specifically, the processes with MPI ranks $(K-1)P_e+1$ to KP_e ($K=1, \dots, M$) are in the K -th global column processor group, and they work on the element E_K at the level of intra-element parallelization. Analogously, the i -th global row processor group consists of the processes with MPI ranks $i, P_e+i, \dots, (M-1)P_e+i$ ($i=1, \dots, P_e$). Any process in the global process grid can be referred to by its row and column indices (I, K) in the process grid, with $I=0, \dots, P_e-1$ and $K=0, \dots, M-1$.

The two level parallelization scheme within DGDFDFT allows (12) to be evaluated efficiently and in a scalable manner. A given block of vectors X is distributed at the inter-element parallelization level in a manner consistent with the element-wise partition suggested by Eq. (12), and it is further distributed by its columns (i.e., Kohn-Sham states) at the intra-element parallelization level. In other words, for a block of vectors of size $N_b \times N_s$ (with $N_b = \sum_{k=1}^M J_K$, i.e., the total number of basis functions in use and N_s denoting the total number of Kohn-Sham states), the MPI process with row and column indices (I, K) holds a block of size $J_K \times \lfloor \frac{N_s}{P_e} \rfloor$ and evaluates this portion of the result (i.e, left hand side of Eq. (12)). In DGDFDFT, the matrix H^{DG} is stored element-wise, i.e., all P_e processes in a given process grid column assigned to a particular element K store all non-zero blocks of the form $H_{K; K'}^{\text{DG}}$. Hence, for a given process, evaluation of (12) only incurs nearest neighbor communication within each process grid row so that blocks of the form $x_{i_1, \dots, i_2; K'}$ (with i_1, i_2 corresponding to the start and end indices of the block of states that the process is working on, and K' corresponding to its nearest neighbor elements) may be obtained.

The strategy of employing a process grid avoids costly global communication and restricts all communication to individual row and column process grids. Additionally,

the block nearest-neighbor type sparsity structure of the Hamiltonian matrix results in further reduction in communication volume. As demonstrated later, these factors result in a particularly well scaling matrix-vector product routine for DGDFT. In contrast, expressing H^{DG} and the block of vectors to be multiplied as dense matrices and the subsequent direct use of parallel dense linear algebra routines (PBLAS⁶⁰, for example) for carrying out the matrix-vector product operation would have incurred a higher computational cost and also significantly degraded the scalability of the computation.

2. Parallel solution of the subspace problem

The various steps involved in solving the subspace problem all require dense linear algebra operations. For example, given the Chebyshev polynomial filtered block of vectors $\tilde{Y} = p_m(H^{\text{DG}})X$, we need to orthonormalize this block of vectors so as to obtain an orthonormal basis for the (approximate) occupied subspace. We carry out this operation by computing the overlap matrix $S = \tilde{Y}^T \tilde{Y}$, computing the Cholesky factorization of S as $S = U^T U$ and then using the Cholesky factor to solve the equation $\hat{Y}U = \tilde{Y}$. The resulting block of vectors \hat{Y} is then orthonormal. The cost of these operations grows cubically with respect to the number of atoms involved in the simulation. Once the number of occupied states exceeds a few hundred, it becomes necessary to parallelize these operations so as to reduce the computational wall times. We use the parallel dense linear algebra routines in the PBLAS⁶⁰ and ScaLAPACK^{29,30} software libraries to do this.

PBLAS and ScaLAPACK routines employ a two-dimensional block-cyclic data distribution over a process grid for their operations. We will refer to this process grid as the *linear algebra process grid*. Since the performance of some of the required routines (particularly, those involved with eigenvalue computation and Cholesky factorization) tend to stagnate (or sometimes, even deteriorate) quite easily if too many processes are in use, we typically use only the first row of processes of the global process grid to set up the linear algebra process grid for problems of moderate size. Thus, the number of processes in the linear algebra process grid typically equals the number of DG elements in use. As the system size grows bigger, we include additional rows of processors in the global processor grid in the linear algebra process grid to reduce the cost of dense linear algebra operations.

Before the sequence of dense linear algebra operations can be initiated, the vector block that contains the product of $p_m(H^{\text{DG}})$ and X must be redistributed over the linear algebra process grid from its distribution over the DG elements. We have implemented routines for seamlessly inter-converting between a block of vectors distributed over the DG elements to one distributed over the linear algebra process grid, at relatively low communication cost²³. In our experience, this step takes no more than

0.1% of the total time spent in the CheFSI routine, even for the largest systems considered here.

The original Chebyshev filtering method presented in^{43,44} employs a QR factorization or the DGKS algorithm⁶¹ for orthonormalization. Here, we have used the faster (but sometimes less stable) Cholesky factorization method instead. We have found this speeds up the orthonormalization by a factor of 2–3 in most cases with no problematic side effects.

With the orthonormalized and filtered block of vectors \hat{Y} , the next step is to compute the projection of H^{DG} onto the occupied subspace: $\hat{H} = \hat{Y}^T(H^{\text{DG}}\hat{Y})$. This step requires the vector block \hat{Y} distributed over the linear algebra process grid to be redistributed over the DG elements, so that the action of H^{DG} on it can be computed. Once again, this data redistribution step takes no more than 0.1% of the total time spent in the CheFSI routine, even for the largest problems considered here.

After diagonalizing the projected matrix \hat{H} , the resulting block of eigenvectors Q can be used to compute the final results of one CheFSI cycle as $X_{\text{new}} = \hat{Y}Q$. Eq. (8) can now be used to compute the diagonal blocks of the density matrix locally on each element, by using the eigenvector coefficients in X_{new} . The corresponding Ritz values Λ_i can be used for adjusting the polynomial filter bounds as well as computing the Fermi energy and occupation numbers.

3. Alignment of eigenvectors with current basis

What distinguishes DGDFT from traditional Kohn-Sham DFT solvers is the change of the basis set in each SCF cycle. This change has implications for the way we prepare the starting vectors for CheFSI on every SCF step. Conventionally, the input to CheFSI at the i^{th} SCF cycle is chosen to be the approximate invariant subspace computed at the $(i-1)^{\text{th}}$ SCF step^{43,45}. However, since the basis set changes from one SCF cycle to the next in DGDFT, the eigenvector coefficients computed in a given SCF cycle relative to the basis of that cycle are not applicable to subsequent SCF cycles with different bases. In practice, the change of basis from one SCF cycle to the next becomes smaller as self-consistency is approached, however, as we show below, the change is sufficiently large to require explicit accommodation to minimize CheFSI iterations.

Figure 3 shows that even for a simple system containing a few hydrogen atoms, a naive implementation of CheFSI, which simply uses the eigenvector coefficients computed in the previous SCF cycle as the starting guess for the current SCF cycle, fails to converge in even 45 iterations (green curve), whereas a direct diagonalization of H^{DG} via ScaLAPACK results in SCF convergence (blue curve) in less than 20 SCF iterations. To address this problem, we may perform several cycles of CheFSI in every SCF step to compensate for the poor initial approximation provided by the coefficients of the previous SCF

step. This strategy produces results closer to those produced by exact diagonalization in each SCF cycle (black curve.) But repeating the CheFSI cycle multiple times on every SCF step increases the overall computational cost of the method.

Since the basis in a given SCF iteration is distinct from that of the previous, with distinct span, the eigenvectors of the previous SCF iteration cannot be expressed in terms of the basis of the current SCF iteration without approximation. For optimality, we choose the best approximation in the ℓ^2 norm, which by virtue of the orthonormality of the DG basis, is readily obtained by ℓ^2 projection. Specifically, if $X^{(i)}$ denotes an $N_b \times N_s$ block of vector coefficients (where, N_b denotes the total number basis functions in use and N_s denotes the number of Kohn-Sham states) computed by CheFSI on a given SCF step, and V^i denotes an $N_r \times N_b$ block of basis vectors corresponding to the ALBs sampled on an N_r -dimensional real-space grid (consisting of Gauss-Lobatto integration points for example²¹), then the starting point for the CheFSI method on SCF step $i + 1$ is given by:

$$X^{(i+1)} = (V^{i+1})^T V^i X^{(i)}. \quad (13)$$

Since the ALBs and the eigenvector coefficients $X^{(i)}$ and $X^{(i+1)}$ are all distributed DG-element wise (i.e., $X^{(i)}$ for example, is represented as M blocks $X_K^{(i)}$; $K = 1, \dots, M$, stacked column-wise), this becomes:

$$X^{(i+1)} = \sum_{K=1}^M ((V^{i+1})^T V^i)_K (X^{(i)})_K. \quad (14)$$

Further noting that ALBs from different elements are orthogonal to each other due to disjoint supports, we may rewrite the above as:

$$X_K^{(i+1)} = (V_K^{i+1})^T V_K^i (X_K^{(i)}), \quad (15)$$

with V_K^i and V_K^{i+1} denoting the matrix representation of ALBs originating from the element K on SCF steps i and $i + 1$, respectively. Eq. (15) can be evaluated locally on each element by means of two matrix-matrix multiplications. As shown by the red curve in Figure 3, this extra step of re-aligning results in SCF convergence with a rate comparable to that of exact diagonalization.

In all the calculations presented here, this extra step of aligning the wavefunction coefficients was always carried out from SCF step 2 onwards. The overhead due to this step is minimal (typically less than 0.1% of the total time spent on a CheFSI cycle) and does not grow with system size since larger systems employ more elements and the re-alignment calculation is carried out locally on each element.

A flowchart summarizing the various steps involved in the DG-CheFSI method is presented in Figure 4.

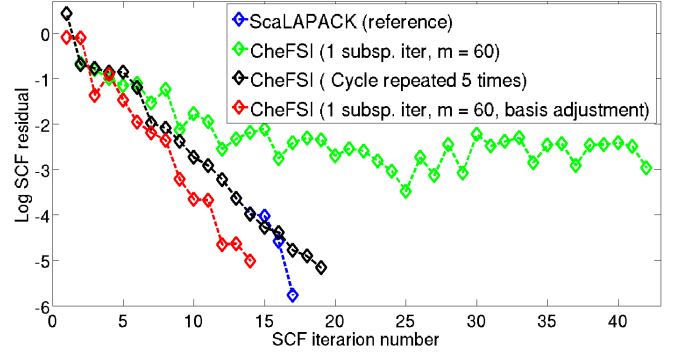


FIG. 3. SCF convergence of normalized electron density residual for different variants of CheFSI within DG-DFT (naive implementation, multiple cycles, and eigenvector re-alignment to adjust for evolving basis set) for a simple system containing a few hydrogen atoms. Reference ScaLAPACK results are also presented.

4. Complexity analysis

For the purpose of this discussion, we let N_b denote the total number basis functions in use (i.e., $N_b = \sum_{K=1}^M J_K$) and we let N_s denotes the number of Kohn-Sham states. Further, we let N_g represent the total number of real-space grid points in use, with N_g/M grid points used for storing each ALB locally within its associated element, with M elements. The quantities $\frac{N_b}{M}$ and $\frac{N_g}{M}$ then correspond to the number of ALBs per element and number of real-space grid points per element, respectively, and are constants for a particular simulation and accuracy level.

As explained above, the CheFSI approach mainly involves the application of the Chebyshev polynomial filter on a block of vectors and subsequent solution of the subspace problem. Within the DG framework, there is an additional step of aligning the DG coefficients of the Kohn-Sham states from one SCF step to the next. Regardless of the basis set in use (e.g., finite elements, planewaves, or ALBs), the subspace problem solution scales as $\mathcal{O}(N_b N_s^2 + N_s^3)$ due to the requirement of dense matrix multiplications⁴³.

Let us now focus on the polynomial filtering step. This involves computing the product of the Hamiltonian matrix with the block of Kohn-Sham states. After the generation process of the ALBs (a step which incurs a memory cost of $\mathcal{O}(N_g N_b/M)$ on every element), the memory cost associated with storage of the coefficients of Kohn-Sham states in terms of the ALBs is $\mathcal{O}\left(N_b N_s + N_b \frac{N_g}{M}\right)$. This contrasts with the storage cost of $\mathcal{O}(N_g N_s)$ that would be required by finite differences, finite elements, or planewave methods using the same number of real-space grid points. In this sense, the use of ALBs can be seen as a systematically improvable compressed format for storing the Kohn-Sham states. In practice, the number of ALBs per element N_b/M is at most a few hundreds and this number is usually far exceeded by the number

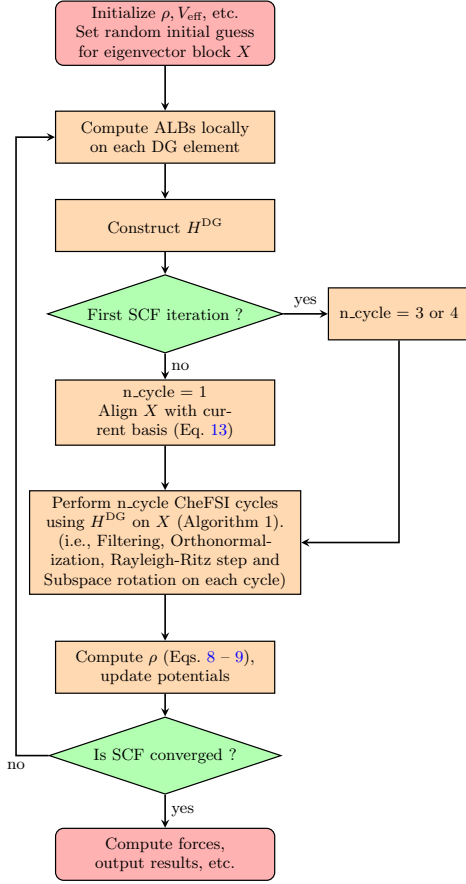


FIG. 4. Flowchart depicting the various steps of CheFSI within DGDFT. Typically, 3 or 4 CheFSI cycles are applied on the first SCF step when starting from a random guess for the eigenvector block X .

of Kohn-Sham states N_s , in large calculations. Further N_b/N_s is typically $2 \sim 20$. Hence, once the ALBs have been generated, there is overall less memory cost involved in storing the Kohn-Sham states using the ALBs.

As explained earlier (section II B 1, Eq. 11), multiplying the block sparse matrix H^{DG} with the block of Kohn-Sham states involves a few dense matrix multiplications of small blocks (of size $\frac{N_b}{M} \times \frac{N_b}{M}$) coming from H^{DG} , with blocks (of size $\frac{N_b}{M} \times N_s$) coming from the coefficients of the Kohn-Sham states, for each DG element. If there are c_N such multiplications to be carried out for each element (this being related to the number of nearest neighbors of elements), the total cost for the application of one step of the polynomial filter is proportional to

$$c_N \left(\frac{N_b}{M} \right)^2 N_s M = \mathcal{O}(N_s M), \quad (16)$$

since $\frac{N_b}{M}$ is a constant. Splitting this calculation into the respective M elements therefore incurs a computational cost of $\mathcal{O}(N_s)$ on every element. Note that, in contrast, this computation using finite differences or finite elements

would have a complexity of $\mathcal{O}(N_s N_g)$ in total and is likely to incur a greater computational cost.

Finally, the step of aligning the Kohn-Sham states with the current basis set on every SCF step (Eq. 14) incurs a cost that is proportional to

$$\left(\frac{N_b}{M} \right)^2 \frac{N_g}{M} + \left(\frac{N_b}{M} \right)^2 N_s = \mathcal{O}(N_s), \quad (17)$$

on every element.

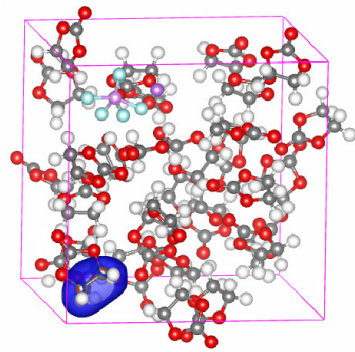
In contrast to the computational complexity of the various steps involved in the CheFSI approach, direct diagonalization of H^{DG} involves a computational complexity of $\mathcal{O}(N_b^3)$ while the PEXSI approach involves a cost of $\mathcal{O}\left(\left(\frac{N_b}{M}\right)^3 M^{\alpha_D}\right)$ (with $\alpha_D = 1.0, 1.5, 2.0$ for one-, two-, and three-dimensional systems, respectively). Direct diagonalization is more computationally intensive while the PEXSI approach results in a larger prefactor, because of which both methods result in longer wall times to solution compared to CheFSI for the full range of system sizes considered here.

III. RESULTS

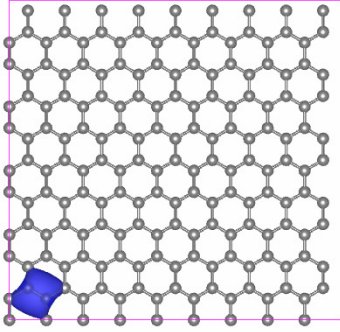
In this section, we investigate the parallel scalability of the CheFSI method and compare its performance with the existing ScaLAPACK and PEXSI methods in DGDFT. Two prototypical systems have been used for our calculations. The first, referred to as *Li3D*, consists of a three-dimensional bulk lithium-ion electrolyte system originating from the design of energy storage devices. Atoms of hydrogen, lithium, carbon, phosphorus, oxygen, and fluorine, numbering 318 in total, are present in this system. The second, referred to as *Graphene2D*, consists of a two-dimensional sheet of graphene containing 180 carbon atoms. These systems were chosen for their technological relevance as well as the fact that KS-DFT calculations on large samples of these systems can be challenging. Figure 5 shows the *Li3D* and *Graphene2D* systems along with the first ALB from one of the DG elements of these systems.

In order to be able to work with larger system sizes, we have employed multiple unit cells of these systems replicated along the coordinate axes. Thus, *Li3D*_{1×2×2} for example, refers to a system in which the 318-atom unit cell has been replicated along Y and Z directions to produce a 1272-atom bulk system; and similarly, *Graphene2D*_{4×4} refers to a graphene sheet containing 2880 atoms.

In what follows, we shall consider the *time to solution*. For the CheFSI and ScaLAPACK diagonalization methods, this will refer to the wall clock time that these methods require to compute the eigenvalues and eigenvectors of H^{DG} as well as the diagonal blocks of the density matrix (via Eq. (8)), during a general SCF cycle. For the PEXSI method, it will refer to the wall clock time that is required to compute directly the density matrix corresponding to the sparsity pattern of H^{DG} . In order to



(a) Bulk Li3D system containing 318 atoms.



(b) Graphene2D system containing 180 atoms.

FIG. 5. Prototype 2D and 3D systems used for the computations in this work. Larger sized systems were obtained by periodic replication of these unit cells. Iso-surface of the first ALB from one of the DG elements of these systems is also shown.

have a fair comparison between the methods, it is important to ensure that the three methods show the same convergence rate over multiple SCF cycles. This then allows the comparison between the methods to be carried out with reference to the time to solution for one SCF cycle. Accordingly, we have adjusted the Chebyshev polynomial filter order as well as the various parameters used in PEXSI (such as the number of poles and the number of chemical potential iterations), so that these methods converge at least as fast as the reference ScaLAPACK calculations. Figure 6 shows the convergence of all the three methods for the prototype systems in use here.

For most of the calculations described here, the polynomial filter order used was between 80 and 100. However, these employed relatively hard pseudopotentials⁶²; lower filter orders may be expected to suffice for softer pseudopotentials. Additionally, in practical molecular dynamics and geometry optimization calculations, fewer SCF cycles are likely to be required for the CheFSI method on every electronic relaxation step, since the method will be able to make use of wavefunction extrapolation (with re-alignment to account for the evolving basis set). Thus, the performance of CheFSI in the con-

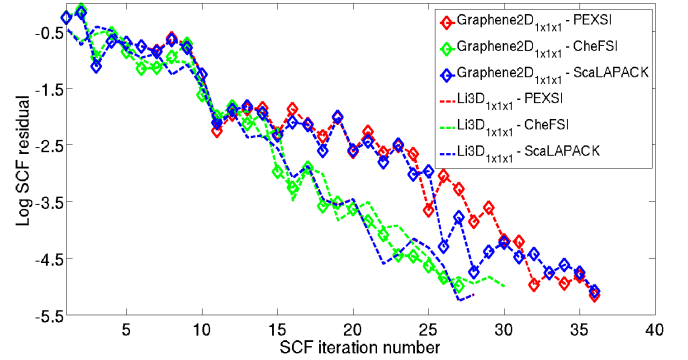


FIG. 6. SCF convergence of normalized electron density residual for two prototypical systems using ScaLAPACK diagonalization, PEXSI, and CheFSI methods.

text of MD simulations may be expected to improve still further relative to the results of static calculations, as presented here.

We have used the local density approximation for the exchange-correlation functional with the parametrization described in⁶³. Hartwigsen-Goedecker-Teter-Hutter pseudopotentials^{62,63} are employed to remove inert core electrons from the computations. We have typically employed 100 – 120 additional states in most calculations to accommodate partial occupation. SCF convergence was accelerated by means of Pulay’s scheme⁶⁴ or its periodic variant⁶⁵, and an electronic temperature of 300 K was used in Fermi-Dirac occupation. To attain chemical accuracy (i.e., error in the total energy less than 10^{-3} Ha/atom relative to the fully converged result; additionally we also ensured that the error in the atomic forces are less than 10^{-3} Ha/Bohr relative to the fully converged result), the 318-atom bulk Li3D system was partitioned into $4 \times 4 \times 4$ elements, with 200 ALBs per element, giving ~ 40 ALBs per atom. Likewise, the 180-atom Graphene2D system was partitioned into $1 \times 6 \times 6$ elements, with 120 ALBs per element, giving 24 ALBs per atom.

All calculations described here were performed on the Edison platform at the National Energy Research Scientific Computing (NERSC) center. Edison has 5462 Cray XC30 nodes. Each node has 64 GB of memory and 24 cores partitioned among two Intel Ivy Bridge processors, running at 2.4GHz. Edison employs a Cray Aries high-speed interconnect with Dragonfly topology for inter-node communication.

A. Scaling performance

We first investigate the strong scaling performance of CheFSI within the DG framework and compare it against that of PEXSI and ScaLAPACK. For this, we consider the systems Li3D_{2×2×2} (2544 atoms, 4536 Kohn-Sham states) and Graphene2D_{6×6} (6480 atoms, 13080 Kohn-

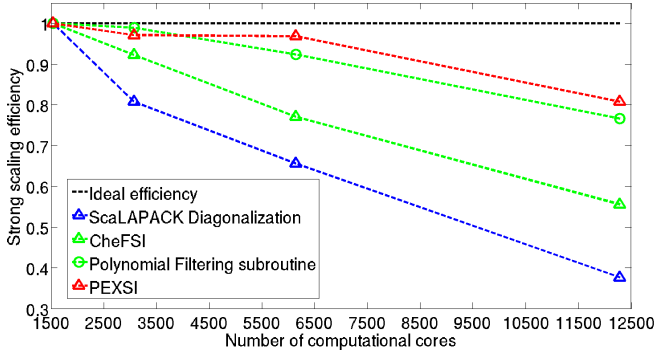
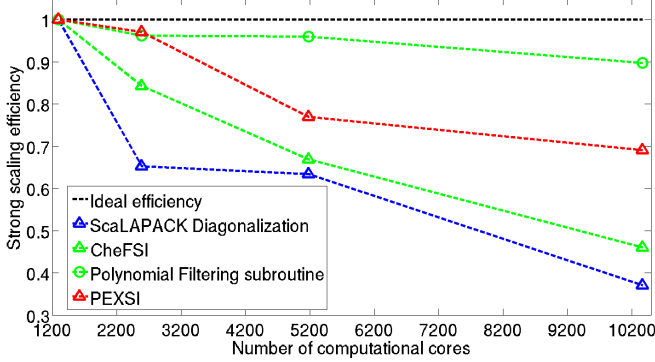
(a) $\text{Li3D}_{2 \times 2 \times 2}$ system (2544 atoms).(b) $\text{Graphene2D}_{6 \times 6}$ system (6480 atoms).

FIG. 7. Strong scaling efficiency of CheFSI in DGDFE, compared against PEXSI and direct ScaLAPACK diagonalization. Scaling performance of the filtering routine is also shown.

Sham states).

Figure 7 shows the wall time to solution (per SCF iteration) vs. number of computational cores employed. From the figures, it is evident that the overall strong scaling performance of CheFSI lies in between that of PEXSI and direct ScaLAPACK diagonalization. For the $\text{Li3D}_{2 \times 2 \times 2}$ system, the performance using 12,288 cores is at about 56 % efficiency (measured against the result from 1500 cores); while for the $\text{Graphene2D}_{6 \times 6}$ system, using 10,368 processors, it is at about 46 % efficiency (measured against the result from 1200 cores). It is interesting to note, however, that the strong scaling performance of the filtering routine by itself is nearly ideal, remaining close to 80 % efficiency for the $\text{Li3D}_{2 \times 2 \times 2}$ case and at about 90 % efficiency for the $\text{Graphene2D}_{6 \times 6}$ case. In particular, the performance of the filtering routine is better in the 2D system due to fewer neighboring elements and correspondingly less communication required. The overall scaling performance of CheFSI, therefore, is limited by the performance of the subspace problem solution, whenever the total time for this step forms a significant fraction of the total CheFSI time. For the systems here, the subspace problem solution time was about 33 % of the total CheFSI time for the $\text{Li3D}_{2 \times 2 \times 2}$ system using 12,288 cores, and 57 % of the total CheFSI time for the

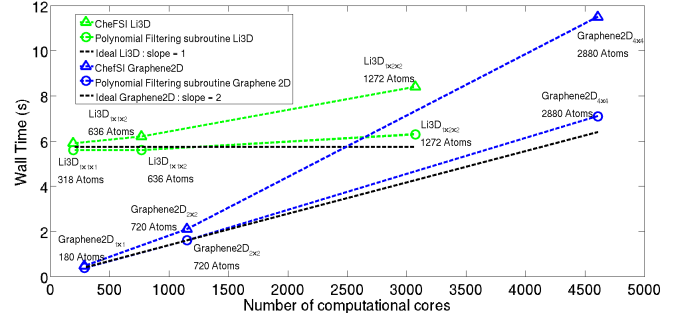


FIG. 8. Weak scaling performance of CheFSI in DGDFE. Performance of the filtering routine is also shown.

$\text{Graphene2D}_{6 \times 6}$ system using 10,368 cores. The larger fraction of time spent on the subspace problem helps explain why the overall scaling performance of CheFSI is somewhat lower for the 2D case here. These observations suggest possible avenues for further improvement of the overall scaling performance of CheFSI.

Next, we investigate the weak scaling performance of CheFSI within DGDFE, i.e., the performance with increasing system size. We investigate the following systems in 3D: $\text{Li3D}_{1 \times 1 \times 1}$, $\text{Li3D}_{1 \times 1 \times 2}$, and $\text{Li3D}_{1 \times 2 \times 2}$. The system sizes have been doubled successively and as a result, the number of Kohn-Sham states involved (approximately) is doubled as well. In 2D, we investigated the systems: $\text{Graphene2D}_{1 \times 1}$, $\text{Graphene2D}_{2 \times 2}$, and $\text{Graphene2D}_{4 \times 4}$. For these cases, the system sizes have been quadrupled successively and as a result, the number of Kohn-Sham states involved (approximately) is quadrupled as well. As a measure of weak scaling performance, the wall clock time to solution (per SCF iteration) is shown in Figure 8. For each system, the number of computational cores was quadrupled successively as sizes were doubled (Li3D) and quadrupled (Graphene2D) successively.

On increasing the system size n fold, the number of DG elements used for the calculation has to increase by the same factor to keep each local calculation manageable and to maintain the same level of accuracy of solution. Since the number of Kohn-Sham states involved also increases n fold, there is an overall n^2 factor increase in the time required for applying the Chebyshev polynomial filter to all the states involved (Eq. 16). Hence, if the number of computational cores used for doing the larger calculation is increased by a factor of s , the expected wall time for applying the Chebyshev polynomial filter will change by a factor of n^2/s . This observation should also hold for the total CheFSI time, as long as the time for solution of the subspace problem forms a small fraction of the total CheFSI time. Thus, for the Li3D systems, the wall-times for each system should ideally remain constant ($n = 2, s = 4$) while for the Graphene2D systems, an increase by a factor of 4 should be observed ($n = 4, s = 4$). Figure 8 shows that this expectation holds reasonably well for the overall CheFSI time, and

particularly well for the Chebyshev polynomial filter application time. For the Li3D systems, the weak scaling efficiency of CheFSI is about 70 % using 3072 cores while it is about 65 % using 4608 cores for the Graphene2D systems. The performance of the polynomial filter application routine for both these systems is close to 90 %. These results demonstrate again the critical importance of an efficient, well scaling subspace solution as system size increases beyond a few thousand atoms.

B. Benchmark calculations

As the final test of computational efficiency, we study the performance of CheFSI on large benchmark systems and compare the wall time to solution between CheFSI, direct ScaLAPACK diagonalization, and PEXSI. We choose the Li3D_{3×3×3} and Graphene2D_{8×8} systems for this study. 13,824 computational cores were used for both systems. The results are shown in Table I.

System	ScaLAPACK	PEXSI	CheFSI
Li3D _{3×3×3}			
8,586 atoms	3323	3784	170
~ 15,000 states			
Graphene2D _{8×8}			
11,520 atoms	2473	426	105
~ 23,200 states			

TABLE I. Solution wall times per SCF step (rounded to nearest second) for direct ScaLAPACK diagonalization, PEXSI, and CheFSI on 13,824 computational cores for two large systems.

The results show that CheFSI is by far the fastest of all the three approaches (up to more than an order of magnitude faster), particularly for the bulk system. Even for the two-dimensional material system, a geometry in which PEXSI is known to perform particularly well, CheFSI is able to outperform with the same number of cores. Due to the good scalability properties of PEXSI, the wall time for the Graphene2D_{8×8} system can be brought down to be comparable to CheFSI (using 55,296 cores for instance), but overall CheFSI remains more economical in terms of computational resources used (total CPU-hours, for example). We have also observed that the timing results remain favorable for CheFSI for smaller systems, such as those used in the scaling performance studies.

In order to obtain an estimate of the SCF wall times achievable with the DGDFT-CheFSI framework on large-scale computational platforms, we studied the Li3D_{3×3×3} and Graphene2D_{8×8} systems using 55,296 computational cores. The results are shown in Table II.

It is apparent from the results in Table II that for these systems, the largest fraction of the total SCF time is spent on the solution of the subspace problem. Thus the

System	ALB Generation	Hamiltonian update	CheFSI (filtering)	Total SCF time
Li3D _{3×3×3}	11	3	76 (36)	90
Graphene2D _{8×8}	5	4	66 (16)	75

TABLE II. Wall times for various stages of the SCF cycle (rounded to nearest second) with the DGDFT-CheFSI approach for two large systems using 55,296 computational cores. The numbers in parentheses indicate the wall time spent on the filtering step.

cubic computational complexity associated with the solution of the subspace problem starts to dominate as the system size grows larger, beyond a few thousand atoms in the present case.

IV. CONCLUSION

We have used Chebyshev polynomial filtered subspace iteration (CheFSI) within the Discontinuous Galerkin method to enable large-scale first principles simulations of a wide variety of materials systems using Density Functional Theory. Due to a number of attractive features of the DG Hamiltonian matrix, the implementation of CheFSI within the Discontinuous Galerkin framework allows the computation of the Kohn-Sham eigenstates of the Hamiltonian to be carried out in a highly efficient and scalable manner. By virtue of the limited spectral width of the DG Hamiltonian matrix, relatively low polynomial orders suffice, reducing the number of matrix-vector multiplies required; while the block-sparse structure of the DG Hamiltonian facilitates efficient, parallel implementation of each multiply. In addition, the strict locality and orthonormality of the adaptive local basis facilitates realignment of eigenvector coefficients from one SCF step to the next, as the basis is optimized on-the-fly at each step. Taken together, these advantages yield an accurate, systematically improvable electronic structure method, applicable to metals and insulators alike, capable of simulating thousands of atoms in tens of seconds per SCF iteration on large-scale parallel computers.

In the near future, we aim to carry out large-scale quantum molecular dynamics simulations of various materials systems using the DGDFT-CheFSI technique. Of particular interest to us are accurate simulations of the solid-electrolyte interphase (SEI) layer in lithium-ion batteries, and we anticipate that the new methodology will enable accurate simulations of unprecedented size.

While the current methodology can simulate a few thousand atoms in a few tens of seconds per SCF iteration with planewave accuracy, to reach further still, to 10,000 atoms or more with comparable efficiency, will require a substantially more efficient and scalable solution of the subspace problem. We aim to address this issue in future work. One possible avenue for making this step more scalable is to replace the use of Cholesky factorization and eigensolution (for the Rayleigh-Ritz step) with operations which involve only parallel dense matrix

multiplications in the occupied subspace. Parallel dense matrix multiplication tends to scale more favorably and therefore stands to relieve the scalability bottleneck in the current approach. Yet another, more radical possibility would be to dispense with the CheFSI methodology completely, thus avoiding the Rayleigh-Ritz step. Computational techniques such as FEAST⁶⁶ or spectrum slicing⁶⁷ might be used to compute the spectrum of H^{DG} instead. However, compared to more conventional methods like CheFSI, these techniques are likely more suitable for the next generation of computing platforms⁴⁹. An interesting avenue for future work, therefore, would be to investigate whether such techniques can be made to yield significant performance benefits on current parallel computing platforms for physical systems of the types and sizes considered here.

Finally, comparison of the performance of DGDFT-CheFSI with other massively parallel electronic structure codes, such as Qbox^{68,69} for example, is another interesting avenue for research, which the authors are pursuing presently.

ACKNOWLEDGMENTS

This work was performed, in part, under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Support for this work was provided through Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research and Basic Energy Sciences (A.S.B., L.L., W.H., C.Y., and J.E.P.), and by the Center for Applied Mathematics for Energy Research Applications (CAMERA), which is a partnership between Basic Energy Sciences and Advanced Scientific Computing Research at the U.S. Department of Energy (L. L. and C. Y.). The authors thank the National Energy Research Scientific Computing (NERSC) center for making computational resources available to them. A.S.B. would like to thank Meiyue Shao (Lawrence Berkeley Lab) for informative discussions and for his help with improving the presentation of the manuscript. The authors would also like to thank the anonymous reviewers for their comments which helped in improving the manuscript.

¹P. C. Hohenberg and W. Kohn, Physical Review **136**, 864 (1964).

²W. Kohn and L. J. Sham, Physical Review **140**, 1133 (1965).

³R. M. Martin, *Electronic Structure: Basic Theory and Practical Methods*, 1st ed. (Cambridge University Press, 2004).

⁴J. Kohanoff, *Electronic structure calculations for solids and molecules: Theory and computational methods*, 1st ed. (Cambridge University Press, 2006).

⁵M. C. Payne, M. P. Teter, D. C. Allan, T. A. Arias, and J. D. Joannopoulos, Rev. Mod. Phys. **64**, 1045 (1992).

⁶G. Kresse and J. Furthmüller, Physical Review B **54**, 11169 (1996).

⁷J. Pask and P. Sterne, Modelling and Simulation in Materials Science and Engineering **13**, R71 (2005).

⁸E. Tsuchida and M. Tsukada, Physical Review B **52**, 5573 (1995).

⁹H. Chen, X. Dai, X. Gong, L. He, and A. Zhou, Multiscale Modeling & Simulation **12**, 1828 (2014).

¹⁰P. Suryanarayana, V. Gavini, T. Blesgen, K. Bhattacharya, and M. Ortiz, Journal of the Mechanics and Physics of Solids **58**, 256 (2010).

¹¹J. R. Chelikowsky, N. Troullier, K. Wu, and Y. Saad, Phys. Rev. B **50**, 11355 (1994).

¹²J. Chelikowsky, N. Troullier, and Y. Saad, Physical Review Letters **72**, 1240 (1994).

¹³S. Ghosh and P. Suryanarayana, arXiv preprint arXiv:1603.04334 (2016).

¹⁴A. Castro, H. Appel, M. Oliveira, C. Rozzi, X. Andrade, F. Lorenzen, M. Marques, E. Gross, and A. Rubio, Physica Status Solidi (B) Basic Research **243**, 2465 (2006).

¹⁵A. S. Banerjee, R. S. Elliott, and R. D. James, Journal of Computational Physics **287**, 226 (2015).

¹⁶A. S. Banerjee, *Density Functional Methods for Objective Structures: Theory and Simulation Schemes*, Ph.D. thesis, University of Minnesota, Minneapolis (2013).

¹⁷W. Hehre, R. Stewart, and J. Pople, The Journal of Chemical Physics **51**, 2657 (1969).

¹⁸J. C. Slater and G. F. Koster, Phys. Rev. **94**, 1498 (1954).

¹⁹J. Soler, E. Artacho, J. Gale, A. Garcia, J. Junquera, P. Ordejón, and D. Sánchez-Portal, Journal of Physics Condensed Matter **14**, 2745 (2002).

²⁰V. Blum, R. Gehrke, F. Hanke, P. Havu, V. Havu, X. Ren, K. Reuter, and M. Scheffler, Computer Physics Communications **180**, 2175 (2009).

²¹L. Lin, J. Lu, L. Ying, and E. Weinan, Journal of Computational Physics **231**, 2140 (2012).

²²G. Zhang, L. Lin, W. Hu, C. Yang, and J. E. Pask, arXiv preprint arXiv:1510.06489 (2015).

²³W. Hu, L. Lin, and C. Yang, The Journal of chemical physics **143**, 124110 (2015).

²⁴W. Hu, L. Lin, and C. Yang, Physical Chemistry Chemical Physics **17**, 31397 (2015).

²⁵D. N. Arnold, SIAM J. Numer. Anal. **19**, 742 (1982).

²⁶J. Kaye, L. Lin, and C. Yang, Communications in Mathematical Sciences **13**, 1741 (2015).

²⁷L. Lin and B. Stamm, arXiv preprint arXiv:1502.01738 (2015).

²⁸L. Lin and B. Stamm, arXiv preprint arXiv:1603.04456 (2016).

²⁹L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, et al., *ScaLAPACK users' guide*, Vol. 4 (SIAM, 1997).

³⁰J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, in *Applied Parallel Computing Computations in Physics, Chemistry and Engineering Science* (Springer, 1995) pp. 95–106.

³¹S. Goedecker, Reviews of Modern Physics **71**, 1085 (1999).

³²D. Bowler and T. Miyazaki, Reports on Progress in Physics **75**, 036503 (2012).

³³N. D. Hine, P. D. Haynes, A. A. Mostofi, C.-K. Skylaris, and M. C. Payne, Computer Physics Communications **180**, 1041 (2009).

³⁴J. VandeVondele, U. Borstnik, and J. Hutter, Journal of chemical theory and computation **8**, 3565 (2012).

³⁵J.-L. Fattebert and F. Gygi, Physical Review B **73**, 115124 (2006).

³⁶L. Lin, J. Lu, L. Ying, R. Car, and W. E, Comm. Math. Sci. **7**, 755 (2009).

³⁷L. Lin, M. Chen, C. Yang, and L. He, J. Phys.: Condens. Matter **25**, 295501 (2013).

³⁸L. Lin, A. García, G. Huhs, and C. Yang, Journal of Physics: Condensed Matter **26**, 305503 (2014).

³⁹E. R. Davidson, J. Comput. Phys. **17**, 87 (1975).

⁴⁰E. R. Davidson, Comput. Phys. Commun. **53**, 49 (1989).

⁴¹F. Bottin, S. Leroux, A. Knyazev, and G. Zerah, Computational Materials Science **42**, 329 (2008).

⁴²E. Vecharynski, C. Yang, and J. E. Pask, Journal of Computational Physics **290**, 73 (2015).

- ⁴³Y. Zhou, Y. Saad, M. L. Tiago, and J. R. Chelikowsky, *Journal of Computational Physics* **219**, 172 (2006).
- ⁴⁴Y. Zhou, Y. Saad, M. L. Tiago, and J. R. Chelikowsky, *Phys. Rev. E* **74**, 066704 (2006).
- ⁴⁵Y. Zhou, J. R. Chelikowsky, and Y. Saad, *Journal of Computational Physics* **274**, 770 (2014).
- ⁴⁶V. Michaud-Rioux, L. Zhang, and H. Guo, *Journal of Computational Physics* **307**, 593 (2016).
- ⁴⁷P. Motamarri, M. Nowak, K. Leiter, J. Knap, and V. Gavini, *Journal of Computational Physics* **253**, 308 (2013).
- ⁴⁸A. S. Banerjee and P. Suryanarayana, *Journal of the Mechanics and Physics of Solids* **96**, 605 (2016).
- ⁴⁹A. Levitt and M. Torrent, *Comput. Phys. Commun.* **187**, 98 (2015).
- ⁵⁰D. Rappoport and F. Furche, *The Journal of Chemical Physics* **133**, 134105 (2010).
- ⁵¹Y. Cai, Z. Bai, J. E. Pask, and N. Sukumar, *Journal of Computational Physics* **255**, 16 (2013).
- ⁵²L. Kleinman and D. M. Bylander, *Phys. Rev. Lett.* **48**, 1425 (1982).
- ⁵³Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, Revised ed. (SIAM, 2011).
- ⁵⁴M. Gu, in *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, edited by Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst (SIAM, Philadelphia, 2000).
- ⁵⁵U. Stephan, D. A. Drabold, and R. M. Martin, *Physical Review B* **58**, 13472 (1998).
- ⁵⁶S. Baroni and P. Giannozzi, *EPL (Europhysics Letters)* **17**, 547 (1992).
- ⁵⁷F. L. Bauer, *Zeitschrift für angewandte Mathematik und Physik ZAMP* **8**, 214 (1957).
- ⁵⁸H. Rutishauser, *Numerische Mathematik* **13**, 4 (1969).
- ⁵⁹H. Rutishauser, *Numerische Mathematik* **16**, 205 (1970).
- ⁶⁰J. Choi, J. Dongarra, S. Ostrouchov, A. Petit, D. Walker, and R. C. Whaley, in *Applied Parallel Computing Computations in Physics, Chemistry and Engineering Science* (Springer, 1995) pp. 107–114.
- ⁶¹J. W. Daniel, W. B. Gragg, L. Kaufman, and G. Stewart, *Mathematics of Computation* **30**, 772 (1976).
- ⁶²C. Hartwigsen, S. Goedecker, and J. Hutter, *Physical Review B* **58**, 3641 (1998).
- ⁶³S. Goedecker, M. Teter, and J. Hutter, *Physical Review B* **54**, 1703 (1996).
- ⁶⁴P. Pulay, *Chemical Physics Letters* **73**, 393 (1980).
- ⁶⁵A. S. Banerjee, P. Suryanarayana, and J. E. Pask, *Chemical Physics Letters* **647**, 31 (2016).
- ⁶⁶E. Polizzi, *Physical Review B* **79**, 115112 (2009).
- ⁶⁷G. Schofield, J. R. Chelikowsky, and Y. Saad, *Computer Physics Communications* **183**, 497 (2012).
- ⁶⁸F. Gygi, R. K. Yates, J. Lorenz, E. W. Draeger, F. Franchetti, C. W. Ueberhuber, B. R. d. Supinski, S. Kral, J. A. Gunnels, and J. C. Sexton, in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing* (IEEE Computer Society, 2005) p. 24.
- ⁶⁹F. Gygi, *IBM Journal of Research and Development* **52**, 137 (2008).